York University
LE/EECS 3101 B, Fall 2022
Assignment 2

Due Date: October 16th, at 23:59pm

A dreamer is one who can only find his way by moonlight, and his punishment is that he sees the dawn before the rest of the world ...

Oscar Wilde

All problems are written problems; submit your solutions electronically **only via Crowdmark**. You are welcome to discuss the general idea of the problems with other students. However, you must write your answers individually and mention your peers (with whom you discussed the problems) in your solution. Please refer to the course webpage for guidelines on academic integrity.

Problem 1 [Bonus] Max Value SubMatrix [4+3+3 = 10 marks]

Let A be an n by n matrix of integers. A "cross" of A is a formed by a contiguous set of items in row i and a contiguous set of items in a column j of A that both contain $A[i, j]$. The "value" of a cross is the sum of the numbers in the cross. In the example below, items highlighted in red form a cross of value -15 with $i = 2$ and $j = 4$, and items in blue form another cross of value 16 with $i \in \{6, 7, 8\}$ and $j = 7$. Note that, with this definition, a set of items in a single row or column or even a single item form a cross.

Present a *divide-and-conquer* algorithm that returns the maximum value of the crosses in A. Do your best to make your algorithm as fast as possible.

-1	18	-4	3	2	7	11	5
-12	1	14	-6	1	8	-7	-5
-3	28	54	-13	2	17	6	35
2	12	14	-16	1	8	-17	-12
-12	8	-3	3	3	9	13	5
3	28	44	-23	2	17	6	45
13	3	13	-6	1	18	-7	-5
2	12	14	26	1	8	17	-12

(a) Describe your algorithm in English or pseudocode.

(b) What is the running time of your algorithm in the *worst case*? Write the recursive formula for the running time and justify your answer. You may assume n is a power of 2.

Answer: Find the answer at the end of this document

Problem 2 Matrix Multiplication [4+4 = 8 marks]

Justin is a smart programmer who has managed to correctly implement both matrix multiplication algorithms we saw in the class. Due to a performance issue in the code, however, the addition of two matrices of size $n \times n$ takes $\Theta(n^{2.5})$ instead of $\Theta(n^2)$.

(a) Write down the recursive formula for the running time of the simple divide-and-conquer algorithm of Slide 24 in Justin's implementation. State the asymptotic time complexity (using Θ notation) and indicate whether the running time changes because of the bug.

Answer: We can write:

$$T(n) = \begin{cases} 8T(n/2) + \Theta(n^{2.5}) & \text{if } n > 2 \\ c \ \text{ if } \ n = 1 \end{cases}$$

This is Case 1 of Master theorem, and we have $T(n) = \Theta(n^3)$. The running time of the algorithm does not change due to the bug.

(b) Write down the recursive formula for the running time of the Strassen algorithm of Slide 24 in Justin's implementation. State the asymptotic time complexity (using Θ notation) and indicate whether the running time changes because of the bug. **Answer:** We can write:

$$T(n) = \begin{cases} 7T(n/2) + \Theta(n^{2.5}) & \text{if } n > 2 \\ d \ \text{ if } \ n = 1 \end{cases}$$

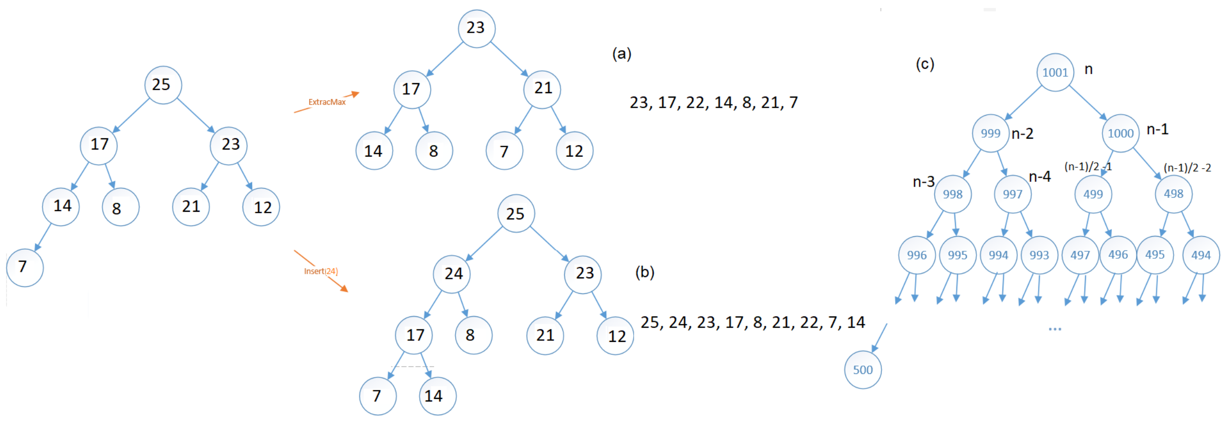
Note that $n^{\log_b a} = n^{\log_2 7} = n^{2.807}$. Again, this is Case 1 of Master theorem, and we have $T(n) = \Theta(n^{807})$. The running time of the algorithm does not change due to the bug.

Problem 3 Heap Operations [3+3+4 = 10 marks]

(a) Consider a heap stored in an array $a = [25, 17, 23, 14, 8, 21, 12, 7]$. Write the updated array when we apply `extractMax` operation.

(b) Consider the same heap $a = [25, 17, 23, 14, 8, 21, 12, 7]$. Write the updated array when we apply the operation `insert(24)`.

(c) Specify the best-case running time of `extractMax()` in a heap formed by n distinct items. If your answer is $\Theta(f(n))$, you must provide an example, with large n, for which extractMax() takes $\Theta(f(n))$. **Answer:** See the following figures. Note that for part (c), extractMax involves placing 500 in the root, swapping it with 1000, and stopping. Therefore, it is true that extractMax takes constant time $\Theta(1)$ in the best case.



Problem 4 Reheap (Heapify) Complexity [2+3+3=8 marks]

Recall that the *height* of a node in a binary tree is the number of edges (links) from the node to the deepest leaf in its subtree (e.g., the height of leaves is 0).

To answer the following question, consider a complete binary tree T of size n in which the last level contains all possible nodes from left to right.

(a) Specify the number of nodes at height h in T for any $h \geq 0$ (no proof is needed).
Answer: The number of leaves in the last level is $(n+1)/2$, and as we "move up", the number of leaves halves at each level. That is, the number of nodes of height h is $(n+1)/2^{h+1}$.

(b) Specify the asymptotic running time of Bubble-Down for a node of height h as a function of h. Then show the total time complexity to Bubble-Down all nodes at level h of T when we apply the Heapify operation on T. Justify your answer in one or two sentences. **Answer:** There are h levels below a node of height h; therefore, Bubble-Down for a node of height h takes $\Theta(h)$. There are $(n+1)/2^{h+1}$ nodes at level h. Thus, the total time complexity for all nodes of level h is $(n+1)/2^{h+1} \times \Theta(h)$.

(c) Use your answer in part to prove that the time complexity of the Heapify operation is $O(n)$. You need to sum the total work over all levels. Show your work.
Hint: $\sum_{x=0}^{\infty} \frac{x}{2^x} = \Theta(1)$.

Answer: There are $\log(n+1)$ levels in T. For the time complexity of Heapify, we can write

$$\begin{aligned}
T(n) &\leq \sum_{h=0}^{\log(n+1)} c \cdot h(n+1)/2^{h+1} \\
&= \frac{c(n+1)}{2} \sum_{h=0}^{\log(n+1)} h/2^h \\
&< \frac{c(n+1)}{2} \sum_{h=0}^{\infty} h/2^h \\
&= \Theta(n)
\end{aligned}$$

Problem 5 Quick-Select [7 marks]

When doing Quick-Select and Quick-Select, it is desired to have a *good* pivot which is almost in the middle of the sorted array. When doing the average-case analysis of Quick-Select, we considered a *good* and a *bad* case; the good case happened when the pivot was among the half middle items of the sorted array, i.e., we had $n/4 \leq i < 3n/4$ (i is the index of pivot in the partitioned array). In our analysis, we provided an upper bound for the time complexity of the algorithm in the *good* case and showed that $T(n) \leq T(3n/4) + cn$ in these cases for some constant c. Since the good case happened with probability 1/2, we could prove that the algorithm runs in linear time on average (see the recursion slide 12 of lectures on selections).

Change the definition of the good case and assume the good case happens when we have $n/3 \leq i < 2n/3$. Provide an upper bound for $T(n)$ and use that to show that Quick-Select runs in $O(n)$.
Hint: start by calculating the probability of good case and bad case happening.

Answer:
We showed in the class that for the average cost of selection algorithm on an array of size n, we have

$$T(n) \leq cn + \frac{1}{n}\left(\sum_{j=0}^{i-1} T(n-j-1) + \sum_{j=i+1}^{n-1} T(j)\right)$$

Assuming $n/3 \leq j < 2n/3$ (when pivot is good, a "good range"), we will have $n-j-1 < 2n/3$ and $j-1 < 2n/3$. Consequently, $T(n-j-1) < T(2n/3)$ and $T(j-1) < T(2n/3)$. With a chance of 1/3, a random pivot is on the left of the good range and with a chance of 1/3, it is on the right of the good range. So, with probability 2/3, the pivot is bad and with probability 1/3, it is good. From the above equation, we get:

$$\begin{aligned} T(n) < cn + 1/3 \ \cdot T(2n/3) + 2/3 \ \cdot T(n) \implies \\ T(n) < 3cn + T(2n/3) \end{aligned}$$

To find the value of $T(n)$, we can use the Master theorem; we are in case 3, $f(n)$ has regularity condition and we have $T(n) = O(n)$.

Problem 6 Median-of-Three Algorithm [4+4=8 marks]

Consider a variant of Median-of-Five algorithm in which, instead of partitioning input into $n/5$ blocks of size 5, we partition the input into $n/3$ blocks of size 3.

a) Follow the same steps as in the slides to derive a recursive formula for the time complexity $T(n)$ of this algorithm.

Answer: Assume x is the selected median-of-medians. So, half of blocks have their medians smaller than x; each of these blocks have one element smaller than their median (and hence smaller than x). So, in total, there are at least $\frac{1}{2} \cdot \underbrace{\frac{n}{3}}_{\text{no. of blocks}} \times 2 = n/3$ items smaller than x. Similarly, there are at least $n/3$ items larger than x. Consequently, the size of recursion can be at most $2n/3$. Assume $T(1) = d$. For $n > 1$, we can write

$$T(n) \leq \underbrace{T(n/3)}_{\text{finding median of meidans}} + \underbrace{cn}_{\textit{selection}} + \underbrace{T(2n/3)}_{\text{size of recursion}}$$

b) Try to solve the recursion by guessing that $T(n) \in O(n)$. Follow the same steps as in the slides and indicate whether we can state $T(n) \in O(n)$.

Answer: Let's guess $T(n) \in O(n)$ and use strong induction to prove it. We should prove there is a value M s.t. $T(n) \leq Mn$ for all $n \geq 1$. For the base we have $T(1) = d \leq M$ as long as $M \geq d$. For any value of n we can state:
$T(n) \leq T(n/3) + T(2n/3) + cn$ (from above recursion)
$\leq M \cdot n/3 + M \cdot 2n/3 + cn$ (induction hypothesis)
$= (M + c)n$

Note that we cannot sow that $(M + c) \leq M$ for any value of M. So, following the same steps does Not give us the same result, i.e., we could Not prove that $T(n) \in O(n)$. [1]

[1] This algorithm's complexity is in fact $O(n \log n)$; remember to use a value ≥ 5 for median of median algorithms.

Answer to Question 1 (sketch)

The "prefixSum" of a row or column is an array in which the ith value is the sum of the elements up to an index i of that row/column.
The "minPrefixSum" an array in which the ith index is the minimum value among the first i entries of the prefixSum.Similarly, the "maxPreSum" is an array in which the ith index is the maximum value among the last $n-i$ entries of the prefixSum.

For example, here are the three arrays for the forth row and forth column of the example matrix in the question (in the matrix, indices start at 1, and the following arrays, they start at 0).
Forth row:
prefixSum: $[0, 2, 14, 28, 12, 13, 21, 4, -8]$
minPrefixSum: $[0, 0, 0, 0, 0, 0, 0, 0, -8]$
maxPrefixSum: $[28, 28, 28, 28, 21, 21, 21, 4, -8]$

Forth column:
prefixSum: $[0, 3, -3, -16, -32, -29, -52, -58, -32]$
minPrefixSum: $[0, 0, -3, -16, -32, -32, -52, -58, -58]$
maxPrefixSum: $[3, 3, -3, -16, -29, -29, -32, -32, -32]$

The value of the continuous subarray with the largest sum in an array A that must contain $A[i]$ is simply $maxPrefixSum[i] - minPrefixSum[i-1]$. For example, in the forth row, if we want to enforce the forth number (with value -16) to be in the contigious subarray, the maximum sum that we get is $21 - 0 = 21$. Similarly, in the forth column, if we want to enforce the same number (-16) to be in the contigious subarray, the maximum sum that we get is $-29 - (-16) = -13$.

We can find the values of minPrefixSum and maxPrefixSum (over rows and columns), using D&Q, as in Algorithm 1. Note that we spend $O(n)$ at each recursive call, and make two recurive calls of size $n/2$. Therefore, the running time of setting minPrefixSum and maxPrefixSum can be described as $T(n) = 2T(n/2) + cn^2$, which is case 3 of Master theorem and solves as $T(n) = \Theta(n^2)$.

Provided with the values of minPrefixSum and maxPrefixSum, we can consider all n^2 possible "centers" for crosses, and find the best possible extension in constant time per each center, as shown in Algorithm 2. Note that we spend a constant time for each center, and the total time complexity is thus $\Theta(n^2)$.

Thus, the total time complexity is $O(n^2)$, which is the best possible (why?)

Algorithm 1 SetValues (A, lo, hi, n)

```
$mid \leftarrow (lo + hi)/2$
$Sum \leftarrow 0$
$minPrefixSumRows[mid][0] \leftarrow 0$
**for** $i = 1$ to $n$ **do**
    $Sum \leftarrow Sum + A[mid][i]$
    $minPrefixSumRows[mid][i] \leftarrow \max\{Sum, minPrefixSumRows[mid][i-1]\}$
$Sum \leftarrow 0$
$minPrefixSumCols[0][mid] \leftarrow 0$
**for** $i = 1$ to $n$ **do**
    $Sum \leftarrow Sum + A[i][mid]$
    $minPrefixSumCols[i][mid] \leftarrow \min\{Sum, minPrefixSumCols[i-1][mid]\}$
$Sum \leftarrow 0$
$maxPrefixSumRows[mid][n+1] \leftarrow 0$
**for** $i = n$ downto $0$ **do**
    $Sum \leftarrow Sum + A[mid][i]$
    $maxPrefixSumRows[mid][i] \leftarrow \max\{Sum, maxPrefixSumRows[mid][i+1]\}$
$Sum \leftarrow 0$
$maxPrefixSumColss[n+1][mid] \leftarrow 0$
**for** $i = n$ downto $0$ **do**
    $Sum \leftarrow Sum + A[i][mid]$
    $maxPrefixSumCols[i][mid] \leftarrow \max\{Sum, maxPrefixSumCols[i+1][mid]\}$
**if** $lo < hi$ **then**
    setValues $(A, lo, mid-1, n)$
    setValues $(A, mid+1, hi, n)$
```

Algorithm 2 FindMaxSum (A, n)

```
$best \leftarrow -\infty$
**for** $i = 1$ to $n$ **do**
    **for** $j = 1$ to $n$ **do**
        $rowSum \leftarrow maxPrefixSumRows[i][j] - minPrefixSumRows[i-1][j]$
        $colSum \leftarrow maxPrefixSumCols[i][j] - minPrefixSumCols[i][j-1]$
        $best \leftarrow \min\{best, rowSum + colSum\}$
```